Smart Lenses

Conrad Thiede, Georg Fuchs, and Heidrun Schumann

Institute for Computer Science
University of Rostock
{*firstname.lastname*}*@uni-rostock.de*

Abstract. Focus + context techniques are widely used for the efficient visualization of large data sets. However, the corresponding adaptation of the representation to the task at hand is not trivial, requiring a suitable model of the visualization goal. One type of focus + context technique is the use of lenses, interactive tools that modify the visualization in a locally confined region and can be 'stacked' to create complex filters. In this paper we propose a new approach to intergrate *smart lenses* into the visualization process based on Chi's Data State Reference Model.This allows us to automatically adapt specific aspects of the visualization based on relevant influence factors, without complex task modeling.

1 Motivation

The size of today's typical data sets is increasing steadily, requiring new strategies in exploring the data to gain insight in an efficient manner. For this purpose information visualization is a well-known approach for its ability to support the analysis of huge data sets. Yet the limited screen size does not allow to show the entire data set in full detail as this would inevitably cause visual cluttering [1]. Consequently, we must show the data with different granularity based on its importance with respect to a given user's goal.

A common approach to this end is the concept of *focus + context*. Here, a region of interest (RoI) is defined as focus that encloses the subset of data elements most relevant to the user, shown in full detail. The context provides an overview of the remaining data with reduced fidelity, and serves to maintain user orientation when navigating the data set. However, deciding what data is relevant is by no means a trivial problem. The automatic RoI specification requires a suitable model of the user's task at hand [2]. For this reason, many approaches rely on the interactive selection of the RoI by the user instead, delegating to her the decision what data is relevant.

One representative of such focus + context techniques are the so-called *lenses*. These are filters that are placed interactively by the user and alter the visualization in a locally confined region [3]. Multiple lenses can also be 'stacked' to combine filter effects [3, 4]. Moreover, applying a filter only to a contained RoI is on principle more efficient than applying it globally.

In this paper, we present a concept for the definition of *smart lenses* based on the the Data State Reference Model (DSRM) by Chi [5], understanding lenses as individual operators within this framework. Smart in this context means that no

holistic task model is required. Instead, single lenses can be employed to modify specific aspects of the visualization within the current RoI. We also provide a systematization of the influence factors that guide the selection and adequate parametrization of those lenses together with some motivating examples.

The paper is structured as follows. In Sect.2 the DSRM is briefly introduced and examples for lens operators on its different stages are given. Section 3 discusses our approach for lens definitions in detail, while Sect. 4 gives examples for their application. Sect. 5 gives a summary and some concluding remarks.

2 The Data-State-Reference Model

Our approach is based on Chi's Data State Reference Model (DSRM) [5]. It describes the visualization process as a pipeline of four *stages* (data states), which are sets of data elements with attributes, and *operators* on the data (Fig. 1). The four stages are, in order of traversal:

1. **Raw data**, i.e. the input data elements with attribute values.
2. **Analytical abstractions**, obtained from raw data values e.g. by calculating statistical moments. Data on this stage is sometimes referred to as meta data.
3. **Visual abstractions**, i.e. high-level visualization concepts such as Scatterplots or 3D surface representations (e.g. triangle meshes).
4. **Image data** that is the result of the rendering process, i.e. pixel data.

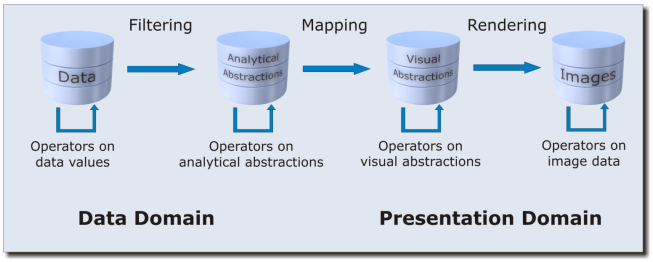

Fig. 1. Data State Reference Model [5] forming the framework for lens integration.

Data is transformed and propagated through the pipeline by operators of two different types, *stage operators* and *transformation operators* [5]. Stage operators work within a single data stage, while transformation operators transfer data from one state to another (cf. Fig. 1). The DSRM further distinguishes between three different types of transformation operators based on the pipeline stages they connect:

- **Filtering** operators transform raw data values into analytical abstractions. This for example includes interpolation of missing values, removal of erroneous readings, and the calculation of mean and extreme values etc.
- **Mapping** operators take analytical abstractions as input and map these to visual abstractions, e.g. by means of color mapping.

– **Rendering** operators process the high-level visual abstractions in order to obtain the visual representation.

In this context, we can also understand lenses as operators in the DSRM, albeit spatially confined to the current RoI [4]. Furthermore, the data states can be divided into two groups [5]. The first two states define the (abstract) data domain, whereas the last two describe data in the presentation domain (Fig. 1). This corresponds to the distinction between semantic and graphical lenses [6]. The versatility of this operator-based concept for lens definitions is illustrated by the following examples.

– Lenses as *stage operators* on *data values* can control the ratio of visualized values in the lens region to avoid clutter (sampling, e.g. [1, 7]), remove errors or interpolate new data.
– Lenses as operators on *analytical abstractions* can be used to sample the corresponding data elements on this stage, as well as to show additional informations about the data, e.g. characteristics of the cluster(s) in the RoI.
– Lenses as operators on *visual abstractions* can for example change the Level-of-detail (LoD) of a given surface mesh [8, 9] or generalize (simplify, aggregate) shapes in the lens region [10].
– Lenses as operators on *image data* realize standard per-pixel image modifications like hue or contrast adjustment, which e.g. can be used to overcome color vision deficiencies, or pixel-based distortions [11].
– Lenses as *filter operators* offer the possibility to calculate additional data characteristics on demand, like node metrics in a large graph [12], and vice versa, to filter out unimportant information like crossing edges to reduce clutter [13].
– Lens as *mapping operators* modify the mapping process and thereby decide about the type of visual representation. This can be used to visualize additional or different other aspects of the data in the RoI than in the remaining image, e.g. [8, 3, 14]
– Lenses as *rendering operators* realize graphical lenses, for example distortion lenses like Rase's Cartographic Lens [15, 16].

Furthermore there are lens functions that consist of more than one operator on multiple pipeline stages, if for example the modification of the visual representation (3^{rd} stage) is done based on semantic information from the data domain (1^{st} or 2^{nd} stages). A representative for this kind of functions is the Semantic Depth-of-Field approach by Hauser et al. [17].

As the examples show, lenses as operators can be applied on every stage of the model in a useful way. In doing so, there are basically two ways to integrate a lens into the visualization process [1]:

– Two separate pipelines are used, one for creating the underlying visualization and one for the lens region.
– The entire image including the lens region is produced using a single pipeline. The lens function alters data on the different stages directly (cf. [1]).

Although with the first approach it is easier to modularize it can not be used for stacking lenses, as their pipelines process the data independently unless elaborate interprocess communication is used. Therefore, we opted to implement a single visualization pipeline that is modified directly by all active lenses [4], with each lens defined as a self-contained plug-in using a declarative script (cf. Fig. 2).

Our goal is to extend our existing approach from [4] by further automating aspects of the lens that previously were left to the user. This will make lenses smart in that, based to the user's goal, only specific aspects of the visualization are adapted within the current RoI.

3 Specification of Smart Lenses

3.1 General Lens Definition

A lens is defined by three parameters *position*, *shape* and the *lens function*.

The *position* of a lens specifies the location of the lens in the visual representation. This is either a 2D point on the virtual canvas in case of a 2D visualization, or 3D coordinates in virtual viewspace.

The *shape* defines the boundary of the RoI enclosed by this lens. Usually, the shape is defined through simple geometries such as rectangles, circles and cubes, spheres for the 2D and 3D cases, respectively. However, more complex shapes are also possible, e.g. polygons with holes. In conjunction with the lens position, the shape therefore uniquely defines the *region* that is affected by the lens function *in view space* (i.e., an area or volume). As a corollary to this, the lens' region changes every time either its position or shape change. Moreover, due to the functional dependencies imposed by the DSRM operators, the shape always partitions the data elements *of each pipeline stage* into two disjoint subsets: the inside of the lens region contains all data elements affected by the lens function, and the second set contains everything else. A lens shape defined in the view space must therefore be mapped into the lower stages if influenced by a corresponding operator. This is a major issue and will be revisited in Section 3.1.

Lastly, the *lens function* can add, remove and modify data elements on each stage of the DSRM just like any other operator. Hence the lens function specifies how the elements within the lens region are processed.

Usually the user specifies these lens parameters interactively with regard to her interests. Our goal is an automatic parameterization to the extend possible. This means a *smart lens* should be adapted automatically with regard to certain aspects of the task at hand. To this end, we first have to identify relevant aspects that may influence lens parameters. Then, meaningful strategies for the automatic parametrization based on those aspects need to be derived.

3.2 Automatic adaptation of lens parameters

The relevant aspects with respect to an automatic lens parametrization can be broken down into three broad categories:

- **User-related aspects:** In general these aspects are specified by means of a user profile that contains information like the overall proficiency or more fine-grained preferences of the user. For example, the user might prefer a circular rather a rectangular lens shape for a given application.
- **Data-related aspects:** Here the lens is adapted with regard to specific data characteristics, e.g. including values similar to a reference value, or child elements of a selection in a hierarchy, in order to set both position and shape of the lens in a way that interesting elements are contained by the lens region. For static data sets as used in our examples, this can be achieved using appropriate filter conditions[1].
- **Task-related aspects:** These aspects reflect the user goals in the context of the tasks at hand. For localization tasks this can mean the automatic placements of lenses over conceivable regions of interest. For identification or comparison tasks, the lens function could for example automatically apply a locally optimized or a global color scale, respectively [18].

The examples above show that different aspects may influence the lens parameters, and that an automatic adaptation can support the user in exploring her data and solving the tasks at hand.

Now the question is how this automatic adaptation should be realized. For this purpose let us have a look at the automatic adjustment of user interfaces in the context of multiple user interfaces for different devices (see [19]). Here, the adaptation process is done based on different models. (i) user model includes the user-driven aspects, (ii) a task model specifies a hierarchy of compound tasks with subtasks and their (temporal, causal) relationships, describing the structure of the task to be solved, and (iii) a dialog model, often derived from the task model, is used to organize the UI in functional groups and to layout GUI elements in a platform-independent way [20].

Instead of applying such extensive model descriptions we propose a simpler strategy: to describe the aspects relevant for the parametrization of smart lenses within the declaration scripts for the corresponding plug-ins. Currently, we use one such script per lens and user that we extend to contain the following additional information (cf. Fig. 2):

- The preferred lens shape,
- the Region of Interest in the data domain, and
- the required lens functions.

The *preferred lens shape* was chosen to include at least one user-driven aspect. The *RoI specification* was chosen because it is important to consider data- and task driven aspects as well. A RoI in the data domain includes data elements from specific attribute value ranges of interest. This can be expressed by means of suitable filter conditions. For our prototype, we use a notation inspired by the OGC Filter Specification [21]. Its XML syntax allows to express complex nested conditions including spatial, arithmetic and logic constraints (cf. Fig. 2, right). A

[1] For dynamically changing data sets, a far more complex approach would be required.

```
...
<shape type="fixed">
    <circle>
        <center>0 0</center>
        <radius>50</radius>
    </circle>
</shape>
<!-- data-driven position: lens position updated internally by operator class -->
<position type="data-driven"/>

<operator class="lvis.smartLenses.MagicEyeLens">
    <paramDefaults>
        <magnify>2.0</magnify>
        <maxdist>50</maxdist>
    </paramDefaults>
</operator>
<operator class="lvis.smartLenses.RiverIcon"/>
...
```

```
...
<shape type="data-driven"/>
<position type="user-driven"/>
<operator class="lvis.smartLenses.TextureLens">
    <Filter>
        <And>
        <PropertyIsEqualTo>
            <PropertyName>krankheit_nr</PropertyName>
            <Literal>5</Literal>
        </PropertyIsEqualTo>
        <PropertyIsBetween>
            <PropertyName>datum</PropertyName>
            <LowerBoundary>2000-01-01</LowerBoundary>
            <UpperBoundary>2000-12-31</UpperBoundary>
        </PropertyIsBetween>
        </And>
    </Filter>
</operator>
...
```

Fig. 2. Examples of lens definition scripts. On the left is the definition of the lens from Fig. 4(left). The script on the right defines the Texture lens from Fig. 5, including a composite filter to specifiy a RoI in data domain.

task-driven specification of RoI also makes sense, e.g. focusing on specific spatial regions. To maintain the highest degree of flexibility we allow the specification of RoI at every stage of the data state reference model. The *lens function* can be selected from a list of predefined functions (i.e. available as plug-in classes, cf. Fig. 2). These functions can be seen as specific operators at different stages of the data state reference model.

Our approach can be summarized as follows: First the script has to be specified, from which the corresponding operators required in the DSRM for the smart lenses are generated. The script therefore abstracts from the actual implementation of the visualization process, and can be reused or modified for other data sets or other visualization settings. Conceptually, such a script can be auto-generated after relevant aspects have been queried from the user, e.g. using a wizzard-style dialog. Subsequently, the visual output is generated by a processing pipeline realizing the DSRM.

As already mentioned in Sect. 3.1, a major point in the context of operator specification is the definition and mapping of the RoI on the different stages. We will discuss this problem in more detail in the next subsection.

3.3 Description of the lens region on every stage of the DSRM

The script defines lens functions as operators within the DSRM. Thus, we need a valid description of the lens region at every stage of the DSRM. To achieve this goal we use the concept of *half spaces*, which is flexible enough to describe different shapes at different stages of the DSRM. Generally, a half-space is that portion of an n-dimensional space obtained by removing that part lying on one side of an $(n-1)$-dimensional hyperplane. Therefore, an arbitrary region in the n-dimensional (data) space can be defined as the intersection of appropriate half spaces. In the simplest case, this means to partition the data space along each relevant attribute axis twice, at the minimum and the maximum value of interest, respectively.

The challenge is to map such region definitions from one stage onto the other and vice versa. In particular, we have to consider the following two cases:

- The lens region has been specified at the first stage (raw data): This is the simplest case. We use the pipeline to map the lens region onto the following stages.
- The lens region specified at a later stage of the pipeline, but the specified lens function requires operators on previous stages: In this case we additionally need an inverse mechanism, or back-projection, to collect the data elements associated with this region on the earlier stages.

There are two approaches to the inverse mapping required for the second case:

- If for each lens operator an inverse operator exists, those inverse operators can be used to calculate the lens region on earlier stages of the DSRM. Usually, however, such inverse operators are not feasible at all, and in some other cases it is not an easy task to define them.
- Another idea is to use a multi-pass approach to specify the lens region on every stage. During the first pipeline pass, for the second and subsequent stages lookup tables are generated. These contain, for each data element on a given stage, the associated input elements for the transformation operators from the previous stage. During the second pass, data elements contributing to the lens regions on the different stages are processed according to the lens function. Besides requiring two passes, a major disadvantage of this approach is the potential memory requirements for the lookup tables, especially for large data sets.

Although the first alternative is faster and generally has a smaller memory footprint, we use the second approach. We can not assume that all inverse operators are given, and we want to assure getting a valid result in any case.

Up to now we have only considered the definition and mapping of a single lens region. In fact, several lens regions at different stages can be defined by scripts. If these lens regions overlap, it will be necessary to detect and solve potential conflicts.

3.4 Combination of Smart Lenses

Combining two (or more) *overlapping* lenses requires two steps. First, the lens regions have to be merged. Since we define lens regions as an intersection of half spaces, their unification is a trivial task.

The second step is the combined execution of the DSRM operators constituting both lens functions of the overlapping regions. This requires an additional effort. There are different approaches imaginable:

- Precedence: Only one lens function is applied to the overlapping region. Which function takes precedence can be determined by the order of specification or priority values from the script.

- Fusion of the lens functions: In this case a new lens function for the overlapping part is specified based on the operators constituting the original lens function. This requires a suitable and consistent function description, e.g. with predicate logic.
- Combination of function outputs: Both lens functions are independently applied to the overlapping region, the output element sets are merged (cf. [1]).

Because a description of the lens function using predicate logic is not always available we deepen the more general approach of combining lens function results.

Furthermore, if we combine the output elements of several functions there can be distinguished three different kinds of potential conflicts:

- The results of two lenses overwrite attributes of the same data elements of the data stage, referred to as a *write conflict*.
- One lens L_1 operates on data elements which attributes are modified by another lens L_2's output. In this case lens L_1 depends on L_2. This kind of conflict thus is a *dependency conflict*.
- If there are two or more lenses which have a potential *write conflict* between their results and a third lens uses these contested data elements as input, a *read conflict* occurs, i.e. an ambiguity about which of the two possible result sets the third lens operator should use.

Based on the assumption that the visualization pipeline itself (i.e. the visualization without applied lenses) is void of conflicts, a single lens will never result in a conflict as lens functions by design always take precedence over regular pipeline operators. Furthermore, read conflicts can only occur if there are unresolved write conflicts. Solving the triggering write conflict automatically cancels the read conflict as well. Dependency conflicts can also be solved easily by appropriately marshaling the execution order of lens operators. Only write conflicts can not be solved by intrinsic means.

However, before conflicts can be resolved they first need to be detected. To this end, formal description of the output elements of the lens operators is required. We chose to use a mathematical set notation as follows.

Each of the four stages in the DSRM is represented by a set A_i of attributes a_{ik}; $1 \leq i \leq 4; 1 \leq k \leq |A_i|$ that are associated with the data elements at the corresponding stage [4], i.e. A_1 correspond to *Raw data* attributes, whereas elements from A_2 represent attributes of *analytical abstractions*, and so on.

A lens operator f transforms data elements with associated attributes from a source set S_n^f (n^{th} stage) to a target set T_m^f (m^{th} stage):

$$S_n^f \subseteq \bigcup_{1 \leq i \leq n} A_i, \quad T_m^f \subseteq \bigcup_{m \leq i \leq 4} A_i$$

This formalism allows us to detect, and in some cases to automatically resolve, the aforementioned conflict types. Let

$$f : S_n^f \rightarrow T_m^f; 1 \leq n \leq m \leq 4, \text{ and}$$

$$g : S_p^g \rightarrow T_q^g; 1 \leq p \leq q \leq 4.$$

A potential *dependency conflict* occurs if $T_m^f \cap S_p^g \neq \emptyset$. In this case g depends on f and therefore, to avoid dependency conflicts f must be executed before g. This guarantees attributes are only used as elements of an input set after all modifying operations have completed.

A *write conflict*, on the other hand, occurs if $T_m^f \cap T_q^g \neq \emptyset$. As mentioned above, although this type of conflict can be detected, it does not lend itself to an inherent solution.

However, smart lenses must have the ability to resolve even write conflicts automatically. One approach to address this problem is a priority mechanism that determines which lens operator takes precedence. To this end, we have extended our script in such a way that it includes priority values for operators.

4 Realization and Usage Examples

Given such a script that reflects the user's interests, we now want to illustrate the different aspects of smart lenses in the scope of the visualization system LandVis [22] for healthcare data. It shows a map divided into areas correlating to the administrative districts on various levels (federal states, districts, ZIP code areas). Different choropleth representations and icon techniques are available for the incident analysis of several diseases (e.g. flu or respiratory problems). The application of the major influence factors identified in Section 3.2 is demonstrated in each of the following examples. The goal in every case is to provide additional information about the data in the region of interest whereas the smart lens application provides the automatic adjustment of relevant parameters.

A common task when exploring geo-referenced data sets is the localization of extreme values. Moreover, the spatial location of this value can exhibit temporal dependecies – for LandVis, monthly aggregates of disease incidents have been recorded over several years –, so one may look for the *current* maximum at the current time step. Figure 3 shows a 'maximum lens' which could be used in this situation. The *lens function* modifies the fill color of the map region according to the number of incidents by superimposing a full red color with increasing opacity. The *lens shape* can be either static according to the user's preference, e.g. rectangular (Fig. 3 left), or data-driven, i.e. adjust itself automatically to match the shape of the currently focussed map area (Fig. 3 right). The (initial) *lens position* is centered on the map area exhibiting current maximum value. Position updates can be data-driven as well, by automatically re-locating the lens when the current maximum changes.

A second common task is to observe a specific region of interest in high detail. An example is the local region where a physician's office is located and where therefore the temporal developement of treated diseases is of primary interest. In this case, most lens parameters will be determined in the script according to user requirements. Figure 4 gives an example for such a lens. The lens is initially positioned over the physician's stated location (address), while the map region

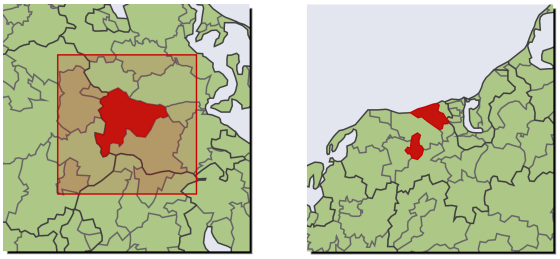

Fig. 3. Example 1: A 'maximum lens' modifies the fill color of map areas according to the number of incidents. The shape is either set to a specific shape (left), or data-driven (right), positioned automatically over the area with the current maximum.

under the lens is also enlarged using a fisheye magnification [15] with a user-specified magnification factor. The lens function was set to use a ThemeRiver [23] icon for temporal analysis.

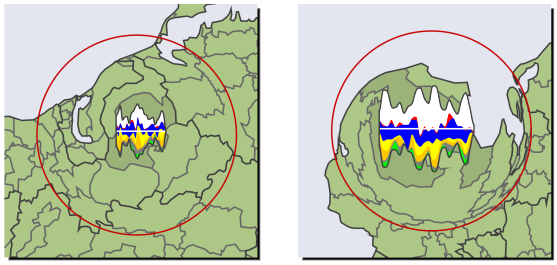

Fig. 4. Example 2: Lens that embeds ThemeRiver [23] icons into the map representation for detail analysis, combined with a fisheye distortion with a user-defined magnification factor. The lens is initially positioned over the user's state 'home' location.

As a last example, Figure 5 shows the support for an analysis of temporal trends in the disease data set. The lens function creates a choropleth representation in the lens region that uses a directional texture to indicate the yearly maximum of incidents for a given disease. Again, the lens shape can either be set to a fixed shape (user-driven, Fig. 5 left), or automatically adapt to the shape of the currently focussed map area (data-driven, Fig. 5 right). The right figure also shows an example of a two-lens combination, where a fisheye distortion enlarges small map regions to enhance the legibility of the directional texture. The lens position in both cases is manipulated exclusively through user interaction as the current region of interest can not be derived from the broad task specification.

5 Conclusion

In this paper, we proposed a new approach to define smart lenses, meaning that our approach allows the versatile definition and combination of arbitray lens

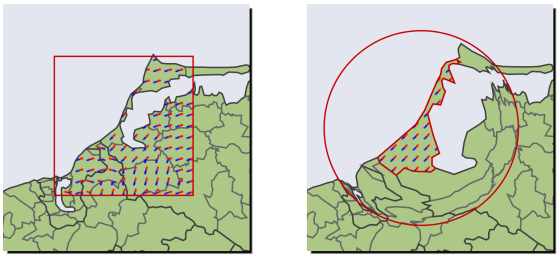

Fig. 5. Example 3: Interactively positioned lens for the analysis of temporal trends. The directional texture in this example consists of red-tipped indicators that point to the month of the yearly incident maximum (1 o'clock is January, 12 o'clock is December).

functions for arbitrarily shaped regions of interest. This is achieved by using as the basis for lens plug-ins Chi's Data State reference Model, wherein lenses can be defined as operators on any stage of the DSRM. Lens regions are defined by means of half space intersection both within the data and the presentation domain. Possible conflicts of lens combinations are detected and partially resolved using a formal set notation.

Thus, single lenses can be employed to modify specific aspects of the visualization within the current RoI. To this end we proposed a systematization of the influence factors that guide the selection and adequate parametrization of those lenses. A major benefit of our approach therefore is that no extensive task model is required.

However, our case studies are only the first step. A user evaluation could provide valueable feedback on the relevance of the aspects considered and their effectiveness in improving the use of lenses. Also, the current solution could be improved in a number of ways. So far, the script-based definition of lenses is limited, especially regarding the operators constituting the lens function. An extension of the script mechanism to allow more flexibility and fine-grained control over operators seems a logical next step. Other aspects that could be improved are the lens prioritization used to resolve write conflicts and the automatic selection of scripts/lens functions based on the current task. So far, we use verbal descriptions of common goals (e.g. localization, identification, comparison, recognition of trends) in the script selection. However, a task model represents a structured task description, and therefore could further enhance the script selection process. Such task models have been used by our research group for the task-specific adaptation of visual interfaces on different platforms [19]. It would be worth to investigate how these task models can be used for automatic adaptation of smart lenses.

References

1. Ellis, G., Dix, A.: Enabling Automatic Clutter Reduction in Parallel Coordinate Plots. Visualization and Computer Graphics, IEEE Transactions on **12**(5) (Sept.-Oct. 2006) 717–724

2. Andrienko, N.V., Andrienko, G.L.: Exploratory Analysis of Spatial and Temporal Data – A Systematic Approach. Springer-Verlag (2006)
3. Bier, E., Stone, M., Pier, K., Buxton, W., DeRose, T.: Toolglass and Magic Lenses: the See-Through Interface. Proceedings of the 20th annual conference on Computer graphics and interactive techniques (1993) 73–80
4. Fuchs, G., Thiede, C., Schumann:, H.: Pluggable Lenses for Interactive Visualizations. In: Poster Compendium of InfoVis'07. (October 2007)
5. Chi, E.: A Taxonomy of Visualization Techniques Using the Data State Reference Model. IEEE InfoVis (2000) 69–75
6. Griethe, H., Fuchs, G., Schumann, H.: A Classification Scheme for Lens Technique. In: WSCG Short Papers. (2005) 89–92
7. Ellis, G., Dix, A.: A Taxonomy of Clutter Reduction for Information Visualisation. Visualization and Computer Graphics, IEEE Transactions on **13**(6) (Nov.-Dec. 2007) 1216–1223
8. Keahey, T.: The Generalized Detail-In-Context Problem. In: IEEE InfoVis, Washington, DC, USA, IEEE Computer Society (1998) 44–51
9. Fuchs, G.A., Holst, M., Schumann, H.: 3D Mesh Exploration for Smart Visual Interfaces. In: Proc. Intl. Conference on Visual Information Systems. (2008)
10. IDELIX Software Inc.: Pliable Display Technology White Paper. www.idelix.com
11. Keahey, T.: Area-normalized thematic views. Proceedings of International Cartography Assembly (1999)
12. Loughlin, M., Hughes, J.: An Annotation System for 3D Fluid Flow Visualization. Visualization, 1994., Visualization '94, Proceedings., IEEE Conference on (17-21 Oct 1994) 273–279, CP31
13. Tominski, C., Abello, J., van Ham, F., Schumann, H.: Fisheye Tree Views and Lenses for Graph Visualization. IV (2006) 17–24
14. Bier, E., Stone, M., Pier, K.: Enhanced Illustration Using Magic Lens Filters. Computer Graphics and Applications, IEEE **17**(6) (Nov/Dec 1997) 62–70
15. Rase, W.D.: Fischauge-Projektionen als kartographische Lupen. In: Salzburger Kartographische Materialien. Volume 26. (1997) 115–122
16. Leung, Y.K., Apperley, M.D.: A Review and Taxonomy of Distortion-Oriented Presentation Techniques. ACM Trans. Comput.-Hum. Interact. **1**(2) (1994)
17. Kosara, R., Miksch, S., Hauser, H.: Semantic Depth of Field. IEEE InfoVis (2001)
18. Tominski, C., Fuchs, G., Schumann, H.: Task-Driven Color Coding. In: Proc. 12th International Conference Information Visualisation (IV'08), London, IEEE Computer Society (8 - 11 July 2008)
19. Biehl, N., Düsterhöft, A., Forbrig, P., Fuchs, G., Reichart, D., Schumann, H.: Advanced Multi-Modal User Interfaces for Mobile Devices - Integration of Visualization, Speech Interaction and Task Modeling. In: Proceedings 17th IRMA International Conference, Washington D.C., USA (2006) (Short Paper).
20. Rathsack, R., Wolff, A., Forbrig, P.: Using HCI-Patterns with Model-Based Generation of Advanced User Interfaces. In: MDDAUI '06 - Model Driven Development of Advanced User Interfaces. Volume 214. (2006)
21. Open Geospatial Consortium: OpenGIS Filter Encoding Implementation Specification, Version 1.1.0 (final). Open Geospatial Consortium Inc. (2005)
22. Schulze-Wollgast, P., Schumann, H., Tominski, C.: Visual Analysis of Human Health Data. IRMA International Conference (2003)
23. Havre, S., Hetzler, B., Nowell, L.: ThemeRiver: Visualizing theme Changes Over Time. In: IEEE InfoVis, Los Alamitos, IEEE Computer Society (2000) 115–123